In lecture we’re going to cover a few simple extensions of the typed-lambda calculus, from TAPL Chapter 11.

1. Products, records
2. Sums, variants
3. Recursion

Homework 6 covers some extensions from Chapter 11 that we haven’t talked about: ascription and lists.

You should also read Chapter 10, and bring questions about it to the recitation.

We’re skipping Chapter 12.

Plan

Simple Extensions

Products
**Evaluation Rules for Pairs**

\[ \text{v}_1 \rightarrow \text{E-PairBeta}_1 \]
\[ \text{v}_1, \text{v}_2 \rightarrow \text{pair value} \]
\[ \text{t}_1 \rightarrow \text{E-Proj}_1 \]
\[ \text{t}_1, \text{t}_2 \rightarrow \text{E-Proj}_2 \]

**Typing Rules for Pairs**

\[ \text{t}_1 : \text{T}_1 \]
\[ \text{t}_2 : \text{T}_2 \]
\[ \{\text{t}_1, \text{t}_2\} : \text{T}_1 \times \text{T}_2 \]

**Records**

\[ \text{r} \rightarrow \text{record} \]
\[ \text{r} \rightarrow \text{record value} \]
\[ \text{r} \rightarrow \text{terms} \]

**Pairs**

\[ \text{v} \rightarrow \text{values} \]
\[ \text{v} \rightarrow \text{terms} \]
\[ \text{v} \rightarrow \text{pairs} \]
Evaluation rules for records

\[
\begin{align*}
{\text{t} \leftarrow \text{name} : \text{L} \leftarrow \{ \text{L} \}} & \quad \text{case a of} \\
{\text{t} \leftarrow \text{x.firstname} : \text{L} \leftarrow \{ \text{L} \}} & \quad \text{for each} \ t
\end{align*}
\]

Sums motivating example

PhysicalAddr = \{ firstlast : String, addr : String \}
VirtualAddr = \{ name : String, email : String \}
Addr = PhysicalAddr + VirtualAddr

\begin{align*}
\text{getName} & = \text{a : Addr}.
\text{case a of} \\
\text{x.firstlast} & \quad \text{inl} \\
\text{y.name} & \quad \text{inr}
\end{align*}

Typing rules for records

\[
\begin{align*}
(\text{L-Rcd}) & \quad \{ u \cdots \ell \_\ell \_ \in \{ \text{L} \} : \{ \text{L} \} \} \leftarrow \text{L} \\
(\text{E-Rcd}) & \quad \{ u \cdots \ell \_\ell \_ \in \{ \text{L} \} : \{ \text{L} \} \} \leftarrow \text{L}
\end{align*}
\]

Sums
For simplicity, let's choose the third.
- Announce each \texttt{in} and \texttt{inr} with the intended sum type.

Ocaml's solution:
- One sum type (requires generalization to "variants," which we'll see next).
- Give constructor different names and only allow each name to appear in
  one "\texttt{in}" as needed during typechecking.

Possible solutions:
- Fix, we've lost uniqueness of types.
  - If \texttt{in} has type \texttt{T}, then \texttt{inr} it has type \texttt{T} for every \texttt{u}.

Problem:

Sums and Uniqueness of Types

\begin{align*}
& \text{(E-Inl)} \quad \frac{x_1 : T_1 \quad \text{inr} x_1 : T_2}{\text{inl} x_1 : T_1 + T_2} \\
& \text{(E-Inr)} \quad \frac{x_2 : T_2 \quad \text{inl} x_2 : T_1}{\text{inr} x_2 : T_1 + T_2} \\
& \text{(E-Case)} \quad \frac{\text{cases of \texttt{in} of } T_1 \lor \text{cases of } T_2}{\text{cases of } T_1 + T_2} \\
& \text{(E-Cast)} \quad \frac{\text{cases of } T_1 \lor \text{cases of } T_2}{\text{cases of } T_1 + T_2} \\
& x_1 \rightarrow x, \quad x_2 \rightarrow x_1 \\
& \text{sum type}
\end{align*}

\begin{align*}
& \text{inl} x_1 : & \text{inr} x_2 : \\
& \text{inl} : & \text{inr} : \\
& \text{sum type of } \texttt{T}_1 \lor \texttt{T}_2 \text{ (the cases \texttt{in} and \texttt{inr} ensure} \\
& \text{\texttt{T}}_1 + \text{\texttt{T}}_2 \text{ is a disjoint union of } \texttt{T}_1 \text{ and } \texttt{T}_2 \text{ (the cases \texttt{in} and \texttt{inr} ensure))}
\end{align*}
New typing rules

$$\Gamma \vdash t : T$$

$$\Gamma \vdash \text{inl} \: t \: \text{as} \: T_1 + T_2 : T_1 + T_2$$

$$\Gamma \vdash \text{inr} \: t \: \text{as} \: T_1 + T_2 : T_1 + T_2$$

Terms

Tagging (left)

Tagging (right)

Values

Tagged value (left)

Tagged value (right)

Evaluation rules ignore annotations:

$$\Gamma \vdash t_1 \rightarrow t'_1$$

$$\text{case} \ (\text{inl} \ v \: \text{as} \: T_0) \rightarrow \begin{array}{l} \text{inl} \ v \: \text{as} \: T_0 \rightarrow [x_1 \mapsto t_2] \\ \text{inr} \ v \: \text{as} \: T_0 \rightarrow [x_2 \mapsto t_2] \end{array}$$

Variants

Just as we generalized binary products to labeled records, we can generalize binary sums to labeled variants.

$$t_1 \rightarrow t'_1$$

$$\Gamma \vdash t_1 \: \text{as} \: T_2 \rightarrow \text{inl} \: t'_1 \: \text{as} \: T_1$$

$$\Gamma \vdash t_1 \: \text{as} \: T_2 \rightarrow \text{inr} \: t'_1 \: \text{as} \: T_2$$
Example

```
addr = &OpAddress; virtual:VirtualAddr

E-Variant

T: case (<physical:x> as Addr)
```

```
E-Case

T: case (<physical:x> as Addr)
```
But we can extend the system with a (typed) fixed-point operator.

Hence, untyped terms like \texttt{omega} and \texttt{fix} are not typable.

In \texttt{4.4}, all programs terminate (Cf. Chapter 12).

Recursion in \texttt{\lambda}.

---

Enumerations

Weekday = \texttt{\{monday, tuesday, wednesday, thursday, friday\}};

\texttt{nextBusinessDay =}\n
\texttt{\{x : Weekday. casewof <monday=x> | <tuesday=x> | <wednesday=x> | <thursday=x> | <friday=x> | <monday=unit> as Weekday\}.}

Recursion

In \texttt{\lambda}, all programs terminate. (Cf. Chapter 12.) Hence, untyped terms like \texttt{omega} and \texttt{fix} are not typable.

But we can extend the system with a (typed) fixed-point operator...

Recursion

Just like in \texttt{\lambda}.

Enumerations

Weekday = \texttt{\{monday, tuesday, wednesday, thursday, friday\}};

\texttt{nextBusinessDay =}\n
\texttt{\{x : Weekday. casewof <monday=x> | <tuesday=x> | <wednesday=x> | <thursday=x> | <friday=x> | <monday=unit> as Weekday\}.}

Recursion

In \texttt{\lambda}, all programs terminate. (Cf. Chapter 12.) Hence, untyped terms like \texttt{omega} and \texttt{fix} are not typable.

But we can extend the system with a (typed) fixed-point operator...
A more convenient form

\[
\text{letrec } x : \text{T}_{1} = t_{1} \text{ int } 2 \text{ def } = \text{let } x = \text{fix}(x : \text{T}_{1}.t_{1}) \text{ in } t_{2}
\]

\[
\text{letrec } x : \text{T}_{1} = t_{2} \text{ def } = \text{let } x = \text{fix}(x : \text{T}_{1}.t_{2}) \text{ in } t_{2}
\]